Notes

Project01 due tonight

- For 10 minute review meeting, please complete by next Monday at the latest
- Drop by office hours
- This afternoon after 3:30 by appointment (email me)
- Via Zoom appointment (email me)

Blackjack tournament

- Plan is 6:30pm Wednesday Nov 2nd, in G12. Let me know if this doesn't work for you
- Maybe 30 minutes, should be fun

Advanced Topics in Class Design

Fall 2022

Member classes

Inheritance

Overriding Inherited Methods

Protected variables and methods

The protected Modifier

The `protected` modifier allows a **child** class to reference a variable or method in the **parent** class

It provides more encapsulation than `public` visibility, but is not as tightly encapsulated as `private` visibility

A protected variable is also visible to any class in the same package as the parent class

Visibility Revisited

It's important to understand one subtle issue related to inheritance and visibility

All variables and methods of a parent class, even private members, are inherited by its children

As we've mentioned, private members cannot be referenced by name in the child class

However, private members inherited by child classes exist and can be referenced indirectly

Visibility Revisited

Because the parent can refer to the private member, the child can reference it indirectly using its parent's methods

The `super` reference can be used to refer to the parent class, even if no object of the parent exists

```
//********************************************************************
//  FoodItem.java       Author: Lewis/Loftus
//
//  Represents an item of food. Used as the parent of a derived class
//  to demonstrate indirect referencing.
//********************************************************************

public class FoodItem
{
   final private int CALORIES_PER_GRAM = 9;
   private int fatGrams;
   protected int servings;

   //-----------------------------------------------------------------
   //  Sets up this food item with the specified number of fat grams
   //  and number of servings.
   //-----------------------------------------------------------------
   public FoodItem(int numFatGrams, int numServings)
   {
      fatGrams = numFatGrams;
      servings = numServings;
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Computes and returns the number of calories in this food item
   //  due to fat.
   //-----------------------------------------------------------------
   private int calories()
   {
      return fatGrams * CALORIES_PER_GRAM;
   }

   //-----------------------------------------------------------------
   //  Computes and returns the number of fat calories per serving.
   //-----------------------------------------------------------------
   public int caloriesPerServing()
   {
      return (calories() / servings);
   }
}
```

```
//********************************************************************
//  Pizza.java       Author: Lewis/Loftus
//
//  Represents a pizza, which is a food item. Used to demonstrate
//  indirect referencing through inheritance.
//********************************************************************

public class Pizza extends FoodItem
{
   //-----------------------------------------------------------------
   //  Sets up a pizza with the specified amount of fat (assumes
   //  eight servings).
   //-----------------------------------------------------------------
   public Pizza(int fatGrams)
   {
      super(fatGrams, 8);
   }
}
```

```
//*********************************************************************
//  FoodAnalyzer.
//
//  Demonstrates                                    vate members.
//*********************************************************************

public class FoodAnalyzer
{
   //-----------------------------------------------------------------
   //  Instantiates a Pizza object and prints its calories per
   //  serving.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      Pizza special = new Pizza(275);

      System.out.println("Calories per serving: " +
                         special.caloriesPerServing());
   }
}
```

Output

```
Calories per serving: 309
```

The super Reference

Constructors are not inherited, even though they have public visibility

Yet we often want to use the parent's constructor to set up the "parent's part" of the object

The `super` reference can be used to refer to the parent class, and often is used to invoke the parent's constructor

A child's constructor is responsible for calling the parent's constructor

The super Reference

The first line of a child's constructor should use the `super` reference to call the parent's constructor

The `super` reference can also be used to reference other variables and methods defined in the parent's class

An child's overriding method `theFunction()` can call its parent `theFunction()` by calling `"super.theFunction()"`

```
//********************************************************************
//  Book2.java       Author: Lewis/Loftus
//
//  Represents a book. Used as the parent of a derived class to
//  demonstrate inheritance and the use of the super reference.
//********************************************************************

public class Book2
{
   protected int pages;

   //-----------------------------------------------------------------
   //  Constructor: Sets up the book with the specified number of
   //  pages.
   //-----------------------------------------------------------------
   public Book2(int numPages)
   {
      pages = numPages;
   }
```

continue

continue

```
   //---------------------------------------------------------------
   //  Pages mutator.
   //---------------------------------------------------------------
   public void setPages(int numPages)
   {
      pages = numPages;
   }

   //---------------------------------------------------------------
   //  Pages accessor.
   //---------------------------------------------------------------
   public int getPages()
   {
      return pages;
   }
}
```

```
//********************************************************************
//  Dictionary2.java       Author: Lewis/Loftus
//
//  Represents a dictionary, which is a book. Used to demonstrate
//  the use of the super reference.
//********************************************************************

public class Dictionary2 extends Book2
{
   private int definitions;

   //-----------------------------------------------------------------
   //  Constructor: Sets up the dictionary with the specified number
   //  of pages and definitions.
   //-----------------------------------------------------------------
   public Dictionary2(int numPages, int numDefinitions)
   {
      super(numPages);

      definitions = numDefinitions;
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Prints a message using both local and inherited values.
   //-----------------------------------------------------------------
   public double computeRatio()
   {
      return (double) definitions/pages;
   }

   //-----------------------------------------------------------------
   //  Definitions mutator.
   //-----------------------------------------------------------------
   public void setDefinitions(int numDefinitions)
   {
      definitions = numDefinitions;
   }

   //-----------------------------------------------------------------
   //  Definitions accessor.
   //-----------------------------------------------------------------
   public int getDefinitions()
   {
      return definitions;
   }
}
```

```
//*****************                                    *****************
//  Words2.java
//
//  Demonstrates
//*****************                                    *****************

public class Words2
{
   //-----------------------------------------------------------------
   //  Instantiates a derived class and invokes its inherited and
   //  local methods.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      Dictionary2 webster = new Dictionary2(1500, 52500);

      System.out.println("Number of pages: " + webster.getPages());

      System.out.println("Number of definitions: " +
                         webster.getDefinitions());

      System.out.println("Definitions per page: " +
                         webster.computeRatio());
   }
}
```

Output

```
Number of pages: 1500
Number of definitions: 52500
Definitions per page: 35.0
```

Advanced Topics in Class Design

Member classes

Inheritance

Overriding Inherited Methods

Protected variables and methods

class Object

The Object Class

A class called `Object` is defined in the `java.lang` package of the Java standard class library

All classes are derived from the `Object` class

If a class is not explicitly defined to be the child of an existing class, it is assumed to be the child of the `Object` class

Therefore, the `Object` class is the ultimate root of all class hierarchies

Even classes you define yourself

The Object Class

The `Object` class contains a few useful methods, which are inherited by all classes

For example, the `toString` method is defined in the `Object` class

Every time we define the `toString` method, we are actually overriding an inherited definition

The Object Class

The `equals` method of the `Object` class returns true if two references refer to the same object (are "aliases")

We can override `equals` in any class to define equality in some more appropriate way

As we've seen, the `String` class defines the `equals` method to return true if two `String` objects contain the same characters

Advanced Topics in Class Design

Member classes

Inheritance

Overriding Inherited Methods

Protected variables and methods

class Object

static variables and methods

Static Class Member Variables

A `static` variable or member "belongs to the class", not to any particular object.

```
Math.PI
```

A static member is invoked through its class name, not an object name

Determining if a variable should be static is an important design decision

- Do values vary for different objects in the class, or not?
- Changing value of a static variable changes it for all objects in the class!

static member functions only can operate on static (and locally declared) variables, not nonstatic class variables!

Static Methods

```
public class Helper
{
   public static int cube(int num)
   {
      return num * num * num;
   }
}
```

Because it is declared as static, the `cube` method can be invoked through the class name:

```
value = Helper.cube(4);
```

Static Class Members

The order of the modifiers can be interchanged, but by convention visibility modifiers come first: "`public static`"

Recall that the `main` method is static – it is invoked by the Java interpreter without creating an object

```
//********************************************************************
//  Slogan.java       Author: Lewis/Loftus
//
//  Represents a single slogan string.
//********************************************************************

public class Slogan
{
   private String phrase;
   private static int count = 0;

   //-----------------------------------------------------------------
   //  Constructor: Sets up the slogan and counts the number of
   //  instances created.
   //-----------------------------------------------------------------
   public Slogan(String str)
   {
      phrase = str;
      count++;
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Returns this slogan as a string.
   //-----------------------------------------------------------------
   public String toString()
   {
      return phrase;
   }

   //-----------------------------------------------------------------
   //  Returns the number of instances of this class that have been
   //  created.
   //-----------------------------------------------------------------
   public static int getCount()
   {
      return count;
   }
}
```

```
//********************************************************************
//  SloganCounter.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the static modifier.
//********************************************************************

public class SloganCounter
{
   //-----------------------------------------------------------------
   //  Creates several Slogan objects and prints the number of
   //  objects that were created.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      Slogan obj;

      obj = new Slogan("Remember the Alamo.");
      System.out.println(obj);

      obj = new Slogan("Don't Worry. Be Happy.");
      System.out.println(obj);
```

continue

continue

```
        obj = new Slogan ("Live Free or Die.");
        System.out.println (obj);

        obj = new Slogan ("Talk is Cheap.");
        System.out.println (obj);

        obj = new Slogan ("Write Once, Run Anywhere.");
        System.out.println (obj);

        System.out.println();
        System.out.println("Slogans created: " + Slogan.getCount());
    }
}
```

Output

```
Remember the Alamo.
Don't Worry. Be Happy.
Live Free or Die.
Talk is Cheap.
Write Once, Run Anywhere.

Slogans created: 5
```

Advanced Topics in Class Design

Member classes

Inheritance

Overriding Inherited Methods

Protected variables and methods

class Object

static variables and methods

Class relationships

Class Relationships

Classes in a software system can have various types of relationships to each other

Three of the most common relationships:

- Inheritance: A *is-a* B
- Dependency: A *uses* B
- Aggregation: A *has-a* B

Student is-a Person
Inventory uses Math
Student has-a String (name, address, etc)

Lab17

CSVTester

Talk about CSV files, show example
Talk about assignment.
Strategies for success: read assignment line by line. Print out and check off reqts as they are completed. Ask questions where you don't understand something.